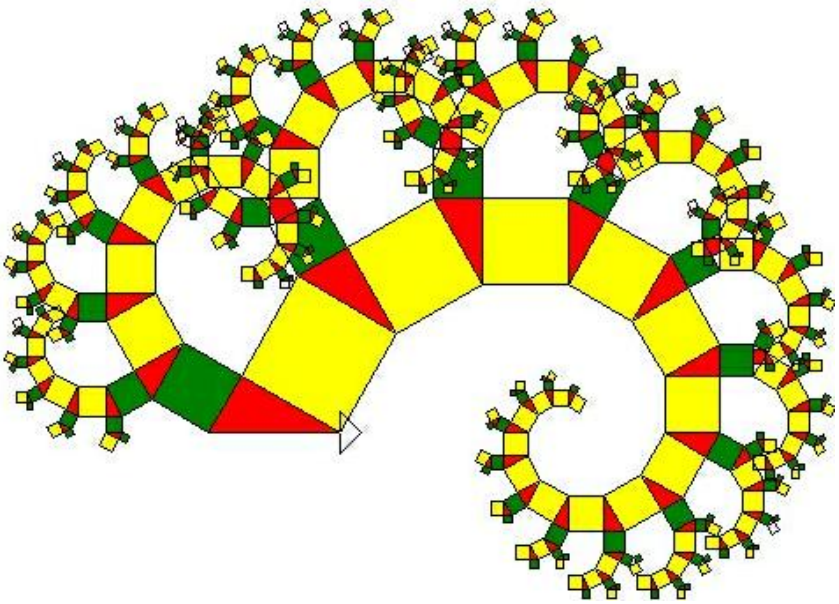
Informatik: Rekursive Programmierung

Ziele

- Mit der rekursiven Programmierung vertraut werden
- Erkennen wo die Unterschiede zwischen rekursiver und iterativer Programmierung liegen
- Die Abarbeitung einer rekursiven Methode verstehen
- Erkennen aus welchen Teilen eine rekursive Methode besteht



Beispiele: Baum des Pythagoras, Fraktale

Ein rekursiver Algorithmus enthält sich selbst als Teil oder ist durch sich selbst definiert

Beispiel: Fakultät n!

$4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$

$n! = n \cdot (n - 1) \cdot (n - 2) \cdot \ldots \cdot 2 \cdot 1$

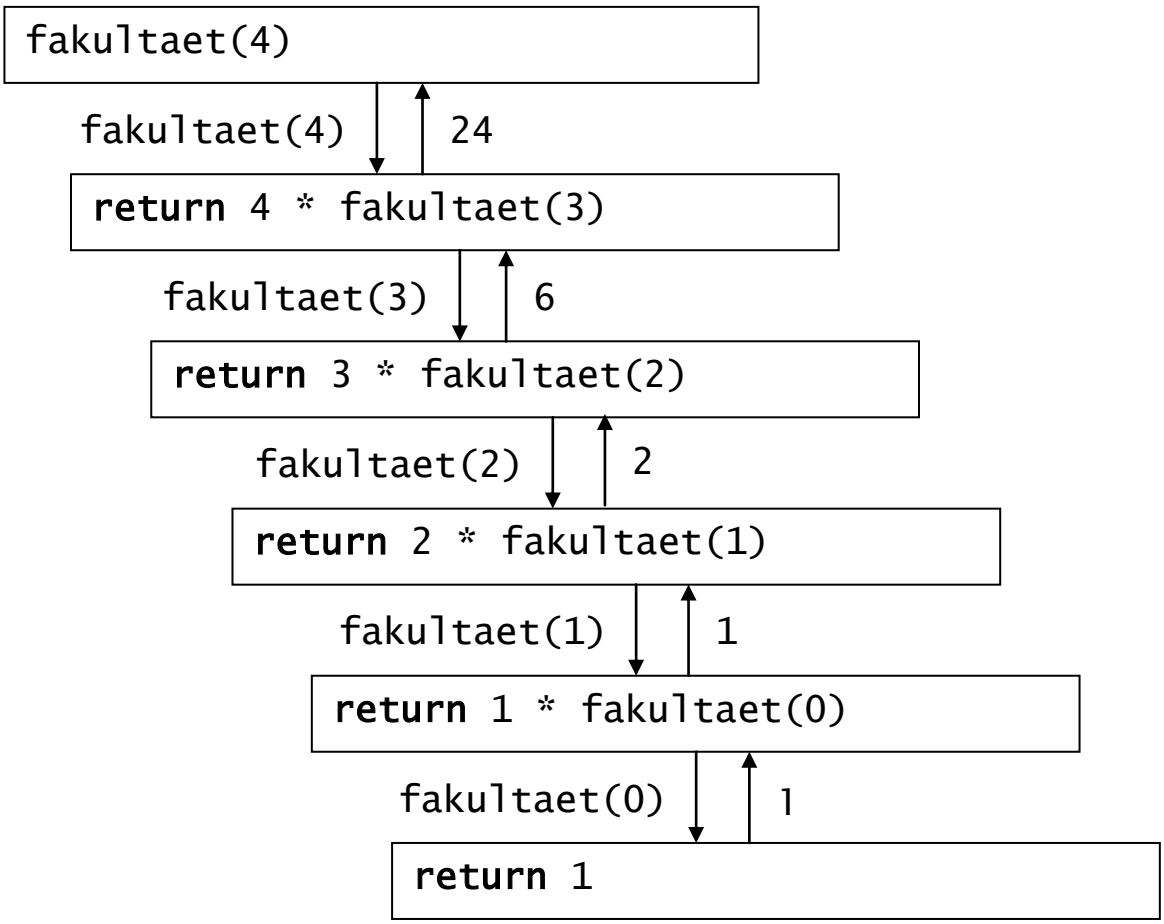
oder

$$n! = \begin{cases} 1 & \text{für } n = 0 \\ n \cdot (n-1)! & \text{für } n > 0 \end{cases}$$

oder

```
public long fakultaet(long n) {
   if (n == 0)
      return 1;
   else
      return n * fakultaet(n - 1);
}
public static void main(String[] args) {
   System.out.println(fakultaet(4));
}
```

Abarbeitung



Eine rekursive Methode besteht aus

1. einleitenden **Abbruchbedingung** (muss irgendwann zutreffen ansonsten *Endlosrekursion*)
2. dem darauf folgenden **rekursiven Aufruf**

Iterative vs. rekursive Algorithmen

```
int n = 10;
for (int i = 0; i <= n; i = i + 1)
   System.out.println(i);

public void printZahl(int start, int end) {
   System.out.println(start);
   if (start < end)
      printZahl(start + 1; end);
}
```

Aufruf: `printZahl(0, 10);`

- Rekursive Algorithmen sind schwer lesbar und brauchen mehr Rechenzeit, weil pro Aufruf im *Programmstapel* Speicher reserviert werden muss
- Es ist möglich, jeden Algorithmus ganz ohne Schleife und lediglich mit Rekursionsaufrufen zu schreiben (*Funktionale Programmiersprachen*)
- Jeder iterative Algorithmus kann auch rekursiv geschrieben werden
- Hat ein rekursiver Algorithmus mehrere rekursive Aufrufe, so kann dieser nur sehr schwer iterativ geschrieben werden

Alles ist Geschmacksache

Beispiel: Fibonacchi-Folge

$$f_n = \begin{cases} 0 & \text{für } n = 0 \\ 1 & \text{für } n = 1 \\ f_{n-1} + f_{n-2} & \text{für } n > 1 \end{cases}$$

f_0	f_1	f_2	f_3	f_4	f_5	f_6
0	1	1	2	3	5	8

Rekursiv

```
public int fib(int n) {
 if (n <= 1)
  return n;
 else
  return fib(n - 1) + fib(n - 2);
}
```

Iterativ

```
int fib = 0;
if (n <= 1)
 fib = n;
else {
 int hilf0 = 0; int hilf1 = 1;
 for (int i = 2; i <= n; i = i + 1) {
  fib = hilf0 + hilf1;
  hilf0 = hilf1;
  hilf1 = fib;
 }
}
```

Beispiel: Modulo

$$\text{mod}(a, b) = \begin{cases} a & \text{falls } a < b \\ \text{mod}(a - b, b) & \text{falls } a \geq b \end{cases}$$

```
public int mod(int a, int b) {
  if (a < b)
    return a;
  else
    return mod(a - b, b);
}
```